**Simple and Inexpensive FPGA-based Coincidence Statistics Acquisition Board**

Technical Description, Installation and Operation Manual

**Abstract**

This document describes how to build and operate a Fast Coincidence Statistics Acquisition Board for hardware detection of coincidences between electrical pulses, such as these generated by Single-Photon Counting Detectors (SPADs). This Board connects to a standard PC via USB-2 cable and can be used in popular data acquisition programs such as LabView.

**Introduction**

The growing interest in research and applications of photon-counting technology is evident. More and more research laboratories use single photon technologies for various applications, such as quantum communication and computing, single-molecule monitoring, precision measurements, etc. Meeting the demand for engineers and researchers with experience in single photon detection and statistical methods requires including simple photon-counting experiments in undergraduate laboratory courses. To implement even a simple photon-counting test bench, one needs to heavily invest in not only SPAD detectors, but also expensive photon counting hardware and software. Even more troubling is the fact that most commercial solutions known to the author provide proprietary (as opposed to open source) software, which makes it difficult to adapt these solutions to custom needs, especially when real-time data processing is needed.

The main principles of development of this board are:

1. Easy assembly, installation and operation
2. Extremely low cost
3. Open Source software and FPGA firmware
4. Immediate connectivity to LabView

These principles provide a fast learning curve with an immediately useful device for simple photon counting (or any pulse counting) applications. At the same time, it allows more advanced users to accommodate unique applications, without spending too much time developing the board-to-PC interface.

**Description**

The board detects “signal” TTL pulses on 4 channels (arrival of their leading (positive) edges) and counts the number of time bins when each of the possible combination of detections has occurred. The time bins are defined by either the board’s own internal clock with 10.42 ns increments (i.e. at 96 MHz) or can be driven by an external clock with up to 10 ns increments (i.e. 100 MHz). The acquired statistics is transmitted via a USB-2 port to the computer, upon computers request. Because only statistical measures are collected, and no individual timestamps are retained, only a few (512 bytes) are transferred each communication cycle. Such low information exchange rate simplifies end user’s data analysis and makes it possible to use slower computers for data acquisition. The number of time stamping channels can be easily increased, by modification of the FPGA firmware.

The board uses an Altera Cyclone II FPGA and a Cypress USB-2 chip. The software was developed and tested on Xylo-EM FPGA development board. It also could be used on a Saxo FPGA development board (with limited functionality). The board is commercially available, costs less than $200 and is the principle expense of the project.

Table 1: Physical Characteristics

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| Computer interface | USB-2 |
| Operating System | Windows 2000, XP |
| Computer Configuration | Any (Tests run on P4 2.8 Mhz, 1 GB ram) |
| Architecture | Open Source |
|  |  |
| Number of data channels | 4 (or more, after firmware modifications) |
| Time stamp increment (internal clock) | 10.42 ns (96 MHz clock) |
| Minimal timestamp increment (external) | <10 ns (>100 MHz clock) |
| Board deadtime | None |
| Event threshold level (detection, clock) | TTL (<1.6 V, positive edge, not adjustable) |
| Highest count rate, combined for all channels | >100 MHz |
| Estimated cost | <$250 |
| Estimated assembly and installation time | 4 h |
|  |  |

Table 2: Parts list

|  |  |
| --- | --- |
| **Part** | **Quantity** |
| Xylo-EM FPGA development board | 1 ea. |
| Generic Electronics Box 3”x2”x6” (min) | 1 ea. |
| USB-2 cable | 1 ea. |
| BNC connectors | 6 ea. (or more) |
| BNC cable | 2 ft. |
| Plastic mounting screws, bolts | As necessary |

**Assembly and installation**

*Assembly*

This manual assumes that Xylo-EM FPGA development board is used. For a different board, changes in firmware and assembly are necessary.

Step 1: Mount the BNC connectors on the box as shown in fig. 1a.

  

Fig. 1. a) mounting BNC connectors; b) mounting the board, top view; c) mounting the board, side view.

Step 2: Mount the FPGA test board to the box as shown in Figs. 1b, c. Make sure that USB connector is accessible from the outside of the box.

Step 3: Solder BNC connectors to FPGA pins with short (3-4”) BNC cable pieces. Make sure that all pieces are of the same length. Refer to table 3.

Table 3: List of BNC inputs

|  |  |
| --- | --- |
| **Pin** | **Purpose** |
| 70 | Detector 1 |
| 73 | Detector 2 |
| 79 | Detector 3 |
| 92 | Detector 4 |
| 88 | External clock |
|  |  |

Note: pins 93, 97, 100, 101 (and many others) could be also wired in the similar way for future extensions and/or debugging.

*Connecting the board*

Note: it is important that all electrical connections are done while the board is not powered up (i.e. not connected to the PC via USB).



Fig 2. Proper termination of used and unused pins.

Step 1: If an external clock will be used, connect an external clock source to pin 88. Connect appropriate input channels to signal sources. It is important to match the impedance to 50 Ohm at least at one of line ends to avoid multiple reflections. In Fig 2, a source of pin 70 is assumed to be a TTL input and is 50 ohm terminated at FPGA. Other sources, such as pins 73 and 88 are terminated at the source. *Note: some commercial detectors produce a digital output of < 3.5 V. If these SPADS are used termination at either end would bring the signal below the threshold voltage. One possible solution is to use extremely short (<1 ft) cables between a SPAD and an FPGA and avoid terminators.* *One can also use simple comparators, set comparators voltage to about 1V and the output at TTL level. One could use a stand-alone FPGA test board that simply translates input to output with a short cable connecting it to a SPAD, and then connect the two FPGA boards via a terminated BNC cable.*

Step 2: Terminate all unused inputs on board, as shown in Fig. 2.

*Software installation*

Step 1: Driver installation. You will need “FX2\_USB.inf” and “FX2\_USB.sys” driver files from the board’s start-up kit.

Connect the board to a computers USB2 port and wait for windows to detect the board and start the Hardware Update Wizard (Fig. 2). Direct the wizard to the directory with “FX2\_USB.inf” and “FX2\_USB.sys” driver files and ignore a warning about windows certification of a driver (if issued). Refer to Fig. 3.



Fig. 3. Step 1 of software installation process.

Step 2: Configuring the board. Open FPGAconf.exe, available from your board’s start-up kit. Select Xylo-EM from menu “Boards” as a target board. Then, in the “Options” menu, set FX2 speed to 48 MHz. Refer to illustration (Fig. 4).

Provide a path to FPGA Time Resolving Acquisition Board firmware: the appropriate .rbf file from this distribution and configure FPGA with the selected .rbf file (click “Configure FPGA” button). The successful completion of this step ensures that the FPGA board is installed correctly and is ready to be used in a Time Resolving Acquisition Board mode.

Step 3: Testing the installation. The simplest way to test the installation is to use the LabView example fpgatest.vi, provided in this package. It can be used with any .rbf firmware. The program tests that electronic events are properly recorded by the board, received by a PC and made available to an end LabView user. The receipt and initial processing of data are handled by a stand-alone dll. This test is successful if the electronic events statistics is correctly displayed by a vi.





Fig. 4. Step 2 of software installation process. Setting up the configuration program.

If the test fails, unplug a USB cable, wait for 10-20 seconds and plug it back in. Repeat steps 2 and 3. If this does not help, the PC needs to be restarted. Repeat steps 2 and 3. If test fails again, check all the connections. It is recommended to use commercial function generators instead of real signal sources for debugging.

**Operation**

There are several default modes of operation. The operation mode can be changed by loading the board with an appropriate pre-compiled firmware (.rbf file). *Using an FPGA configurator, select the .rbf file corresponding to the mode of operation of choice and click “Configure FPGA” button. This procedure has to be done after every off-on power cycle to a board.* The following table for default modes of operation. Advanced users are encouraged to make custom firmware modifications, and share them with the project core, however, custom firmware is not supported by this project.

Table 4: Available operation modes

|  |  |
| --- | --- |
| **Operation mode** | **Firmware file** |
| Internal clock timestamping 96 MHz | internal.rbf |
| External clock timestamping | external.rbf |
|  |  |

Note that because of an external clock capability, this board can easily synchronize to the experiment, and thus reduce timing jitter. In some cases, the synch signal produced by the electronics is not TTL, and therefore has to be converted to TTL.

**LabView interface**

In this section we will discuss basic operation modes using standard existent software from the viewpoint of LabView integration. The basic example of operating the board, fpgatest.vi, covers all important stages of communication, and can be used for various design extensions.

The LabView interface, fpga\_dll.dll is written in C++ and interfaces to LabView vi’s via calls to external dll functions. For convenience and compatibility, these function calls are encapsulated in vi’s and should be called from users programs.

fpga\_init.vi:

This function initializes the communication protocol between the computer and the board. It returns a *handle* to the communication channel to be used for all subsequent operations with the board. On error, it reports an internal driver error number for debugging.

Inputs: none

Outputs: device handle, error message

fpga\_interface.vi:

This function communicates with the board and acquires statistical measures of events (correlations). Please refer to the appropriate section below for file format. Note: this instrument does not acquire timestamps of individual events and therefore does not allow writing a raw file of timestamps to a hard disk. If end user’s application needs this information, use a different instrument (i.e. Fast Time Resolving Acquisition Board).

Inputs:

handle\_in: the output of fpga\_init.vi

fpga\_runs: how many times the dll should query the board before returning the execution to labview. It is recommended to keep this number high enough to ensure that all real-time data makes it to the PC, but low enough so that the custom labview application keeps a comfortable update rate.

takedata: Turns data acquisition on and off

 false: stop taking data

 true: start/continue taking data

Outputs:

handle\_out: “returns” the handle to the system. Can be used for sequential calls to this and other board vi’s.

error: reports on an error encountered during communication (if any)

stats: an array of statistical data about detected events. Refer to the table for the information provided. Note that the maximal number of events that can be stored by the dedicated hardware depends on the length of the counter used. In our design, we assumed that the number of “empty” time bins, when no positive edges are detected is much higher than the number of time bins when any of the “events” happen. Further, we assumed that a 4-fold coincidence is a very rare event, therefore we adjusted the length of hardware counters accordingly (see the table).

Note: Even when data acquisition is switched off, the board accepts and correctly responds to an interface query command. The produced statistics array should normally contain zeros. The very first interface query after acquisition is stopped could return nonzero values in a statistics array. These events have been collected after the last readout but before the stop command was executed.

Table 5: Statistical data provided by fpga\_interface.vi by default

|  |  |  |
| --- | --- | --- |
| **Offset** | **Length (bytes)** | **Meaning** |
| 0 | 6 | Empty time bins |
| 1 | 4 | Single events: Channel 1 |
| 2 | 4 | Single events: Channel 2 |
| 3 | 4 | Single events: Channel 3 |
| 4 | 4 | Single events: Channel 4 |
| 5 | 4 | Coincidence events: Channels 1&2 |
| 6 | 4 | Coincidence events: Channels 1&3 |
| 7 | 4 | Coincidence events: Channels 1&4 |
| 8 | 4 | Coincidence events: Channels 2&3 |
| 9 | 4 | Coincidence events: Channels 2&4 |
| 10 | 4 | Coincidence events: Channels 3&4 |
| 11 | 4 | Coincidence events: Channels 1, 2&3 |
| 12 | 4 | Coincidence events: Channels 1, 2&4 |
| 13 | 4 | Coincidence events: Channels 1, 3&4 |
| 14 | 4 | Coincidence events: Channels 2, 3&4 |
| 15 | 2 | Coincidence events: Channels 1, 2, 3&4 |

Note: Coincidence events are defined as events that occurred during one and the same clock cycle.

fpga\_close.vi:

This function closes the communication protocol, but leaves the mode of the board (either acquisition or idle) unchanged. It is recommended to switch the board to idle by calling an fpga\_interface vi with an appropriate command during the previous communication.

Inputs: handle\_in

Outputs: none

**Further improvements (roadmap)**

1. Extend the number of input signal lines
2. The highest frequency of the test board to clock timestamps is estimated 396 MHz (i.e. 2.52 ns). Device and implement the timestamping routine to approach this frequency.